Шифр роботи: “Логіка”

Галузь науки: Середня освіта (Інформатика)

ОСОБЛИВОСТІ НАВЧАННЯ ІНФОРМАТИКИ ЗАСОБАМИ ОНЛАЙН-СИСТЕМИ У ШКОЛАХ ІЗ ПОГЛИБЛЕНИМ ВИВЧЕННЯМ ІНОЗЕМНОЇ МОВИ

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

JWT – Json Web Token

CDN – Content Delivery Network

SaaS – Software as a service

DDoS – Distributed Denial of Service

XSS – Cross-Site Scripting

UI – User Interface

ВСТУП

У сучасному світі вивчення предмета інформатики в школі є вже необхідністю, адже комп'ютеризація проникла вже практично в усі сфери життєдіяльності людини. Але зараз навчальний процес у більшості закладів здійснюється дистанційно або частково дистанційно, тому з'являється необхідність переходу від контактного спілкування до програмного, починаючи з відеоконференцій і закінчуючи текстовими повідомленнями через месенджери, у зв'язку з чим учасники дистанційного навчання стикаються з проблемою, що різного виду інформації стає більше, а знаходити серед цього потоку більш важливу стає складніше. Тому дана робота присвячена підвищенню ефективності вивченню предмету інформатики за допомогою онлайн-системи, а саме веб-додатку.

Онлайн-системи примножують число користувачів та покращують успішність, але і коштують вони чимало, тому краще створити власну. Перше питання, що виникає перед командою розробників, це проєктування системи. Існує велика кількість технологій, за допомогою яких може бути реалізований веб-додаток (SaaS система). В даній роботі проведено аналіз створення сучасного веб-додатку на прикладі того, що необхідно для вивчення інформатики. Інфраструктура будь-якого сучасного веб-додатку повинна містити в собі рішення щодо реєстрації доменного імені та обробку https запитів, сервіс, що слідкує за балансуванням навантаження, сервіс самого веб-додатку та сервіс, що виконує бізнес-логіку та ін.

Найбільш зручною та популярною системою зв’язку між сервісами є RESTful API, мовою програмування сучасних веб-додатків є JavaScript, для розгортання ж використовують контейнеризацію та послуги хмарних провайдерів.

В даній роботі запропонована архітектура, що може бути використана майже в будь-якій предметній області, з використанням сучасних технологій. Зауважені усі аспекти щодо проєктування та розробки, витримано усі вимоги, без яких у сучасному світі веб-додаток не може вважатися повноцінним.

1. ОГЛЯД ПЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика предметної області та загальних підходів до проблематики теми

Сьогодні інформаційна компетентність в тандемі зі знанням мов - стає провідною складовою технічної підготовки учнів, в якій би сфері діяльності їм не довелося працювати в майбутньому. Отже, переваги іноземного навчання інформатиці стають очевидними [1]:

- розширення методів, що дозволяють учням знаходити інформацію самостійно, а не просто забезпечувати репродуктивне засвоєння знань;

- формування інформаційної компетентності;

- навчання використанню технологій, що забезпечують загальний розвиток особистості;

- адаптація в інформаційному суспільстві.

Сприятливою сферою розширення даного завдання в навчальному процесі є інтеграція уроків англійської мови та інформатики. Ці предмети легко інтегруються так як основна термінологія інформатики має англомовне походження, а «вміння працювати з інформацією відноситься до загальнонавчальних умінь» [1]. Для прикладу були взяті терміни інформатики, що часто зустрічаються (табл. 1.1).

Таблиця 1.1- Приклад термінів інформатики, що часто зустрічаються

Англійський термін	Український еквівалент
Menu	Меню
Byte	Байт
Bit	Біт
Internet	Інтернет
To copy	Копіювати
To format	Форматувати
File	Файл

Однак при інтеграції двох предметів можуть виникатити такі труднощі:

- вчитель повинен володіти мовою на належному рівні (грамотно і доступно викладати предмет);

- вчителю необхідно володіти навичками педагогічного дизайну (ретельне опрацювання матеріалу відповідно до цілей навчання; модифікація навчальних курсів у відповідності з поточними результатами навчання);

- відбір і адаптація матеріалів при підготовці уроку займає набагато більше часу;

- відповідність мовного рівня учнів рівню матеріалу, який видається на уроці.

Ця проблематика вирішується переглядом відео-контенту в оригінальній озвучці від носіїв мови, що застосовується і впроваджується в процес викладання і навчання. При такому підході використання інформаційних технологій на уроках, наприклад, англійської мови сприяє підвищенню мотивації до вивчення мови, так і використання іноземної мови може підвищити інтерес учнів до такого предмету як інформатика, що створить умови для їх успішної самореалізації в майбутньому. Прийшовши на урок, побачивши знайомі слова, фрази, команди, учні будуть більш залучені в навчальний процес, вони не будуть витрачати час на переклад та заучування складних, незнайомих термінів. Широке використання комп'ютерів в житті і в навчанні постійно розширює обсяг і види інформаційної діяльності учнів. Багато з них вже усвідомлюють «"перекачування" трудових ресурсів з матеріальної сфери в інформаційну» [1].

Старшокласники орієнтовані на нові професії, безпосередньо пов'язані з обробкою інформації (WEB-дизайнер, адміністратор локальної комп'ютерної мережі, діловод зі знанням персонального комп'ютера і т.д.). Практично всі з сучасних конкурентноспроможних спеціальностей, спираються на знання розмовної державного та іноземної мови і на вміння використовувати їх в технічному плані (програмне забезпечення, робота з БД та ін).

Отже, курс інформатики, що викладається іноземною мовою, а особливо англійською дозволить отримати учням такі знання, сформує в учнів

призначені для користувача навички, розширить словниковий запас з англійської мови, сформує навички орієнтації в англомовному інтерфейсі прикладних та офісних програм.

Окрім специфіки предметної області, розробники повинні вирішити стандартний набір проблем та досягнути цілей, що стоять перед кожним з таких додатків (перераховано найбільш важливі) [2]:

– цілісність. Стосується бази даних, передбачає коректність даних і їх несуперечливість. Зазвичай вона також включає цілісність зв'язків, що виключає помилки зв'язків між первинним і вторинним ключами;

– чутливість. Люди припускають відсутність візуального зворотного зв'язку для своїх дій тільки протягом дуже обмеженого часу. Для безперервних дій користувача, таких як скролл, відсутність реакції програми можна відновити лише протягом найкоротшого періоду;

– час відгуку. Комп'ютерні обчислення і передача даних по мережі вимагає часу. Це означає, що будь-яка дія, що вимагає нових даних, коду або завантаження додаткових матеріалів, є потенційно асинхронною і має обробляти стан свого завантаження;

– навігація. Дизайн додатку має відповідати UX стандартам та бути інтуїтивно зрозумілим при використанні;

– кешування. Використання кешу даних та регулювання їх актуальності в кожен момент часу є найбільш важливою задачею;

– ентропія. Результат будь-якої послідовності дій користувача повинен бути очікуваним та випливати з дизайну додатку;

– пріоритет. Зі збільшенням масштабу додатки та різні його частини, написані різними людьми або навіть командами, починають змагатися за обмежені ресурси, такі як обчислювальні потужності процесора, трафік мережі, місце на екрані або розмір бандла;

– доступність. Дизайн має враховувати потреби різних категорій населення, виходячи з цільової аудиторії;

– інтернаціоналізація – підтримка декількох з основних мов;

– доставка. Код додатку має бути доставлений користувачеві якомога швидше, навіть, якщо користувач знаходиться на іншому материку. На допомогу приходять хмарні провайдери зі своїми CDN та кешуванням контенту;

– гнучкість. Кожен додаток має бути доступним до масштабування та змін у навколишньому світі.

1.2 Аналіз існуючих онлайн-систем для вивчення інформатики

На різних інтернет-платформах можна знайти мобільні додатки та веб-системи для вивчення інформатики: Tutoronline, «ІнтернетУрок», Наші Пенати, Pixel, МетаШкола, Тетрика, Foxford. Зробимо огляд самих популярних згідно рейтингу онлайн-систем [3] (рис. 1.1).

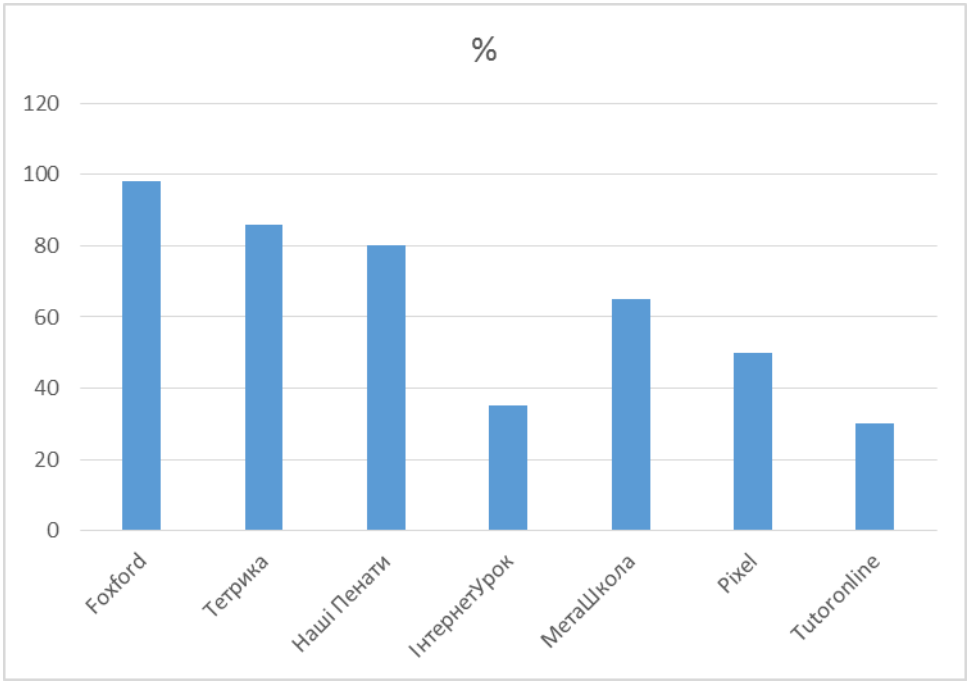

Рисунок 1.1 – Рейтингова оцінка онлайн-систем вивчення інформатики

Перший додаток має назву «Foxford» [3]. Ця система надає можливість вибрати розділи інформатики для навчання, кожен розділ вже має свої приклади. Контент може бути вибраний як з того, що надано адміністраторами так і з введеного користувачами самостійно. Всі заняття проходять на власній

інтернет платформі. Це сучасна інтерактивна дошка, яка адаптована під кожен предмет, онлайн відеозв'язок з викладачем, чат з «однокласниками» і є мобільний додаток. Слід зазначити, що даний додаток користується найбільшим попитом на ринку російськомовних користувачів та нерідко є предметом порад вчителів мови.

Другим конкурентом нашої системи виступатиме додаток «Тетрика» [3]. Він містить в собі адаптований інтерфейс: як для простих завдань, так і для складних (наприклад, побудова математичного графіка), інтерактивну дошку з онлайн чатом, безліч онлайн тренажерів і завдань та можливість надіслати домашнє завдання на перевірку.

Третім серед конкурентів варто виділити сегмент користувачів платформи Наші Пенати, які розміщують відеоуроки, як основний шлях до навчання інформатики [3]. У цей спосіб, система має значно меншу кількість споживачів через відсутність різноманітності методів навчання.

Всі інші онлайн-системи не розроблені на достатньо високому рівні та більшість з них мають один значний недолік - неможливість вивчати інформатику на англійській мові. Також є системи, що нагадують книги, де навчання розподілено за певними розділами та відсутній рівень взаємодії з користувачем.

Отже, проаналізувавши можливих конкурентів системи, ми можемо класифікувати наш продукт як поліпшений варіант вивчення інформатики. Серед вимог до даного продукту поставлена задача вирішення вже існуючих проблем у конкурентів, реалізація їх кращих сторін та внесення деяких принципово нових функцій.

2. ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОНЛАЙН-СИСТЕМИ ВИВЧЕННЯ ІНФОРМАТИКИ

2.1. Постановка задачі та вибір технологій для реалізації програмного забезпечення

В системі повинні бути враховані наступні типи користувачів:

– адміністратор;

– користувач (може як вивчати мову так і виступати постачальником контенту).

Учень вивчатиме інформатику через перегляд відео-контенту в оригінальній озвучці від носіїв мови, одночасно маючи онлайн відеозв'язок з викладачем та чат з «однокласниками». Також, буде інша група користувачів – перекладачі. Саме вони постачатимуть переклад сетів слів, що відносяться до конкретного відео. Ці набори мають бути розподілені стосовно рівня знань тих, хто вивчає іноземну мову. Навчання проходить у наступний спосіб: учень вивчає сет слів, після чого переглядає відеоматеріал та надсилає результат (проходить тест щодо переглянутого матеріалу) до системи, на основі чого формується звіт про успішність користувача. На основі успіхів у навчанні видається більш складний контент та повторення матеріалу.

Серед задач адміністратора можна виділити такі, як: додавання нового контенту, редагування будь-яких даних та їх видалення, керування правами користувачів та зміна основних елементів інтерфейсу (за потреби).

Додаток має відповідати всім вимогам, що висунуті до розробки сучасних веб-додатків (див. п. 1.1).

Для досягнення мети необхідно розв’язати наступні задачі:

– проаналізувати предметну область;

– спроєктувати модель даних для БД типу NoSQL;

– розробити дизайн додатку та реалізувати сервіс, що відповідатиме за видачу контенту;

– розробити сервіс API, що відповідатиме за виконання бізнес-логіки та роботу з базою даних;

– розробити сервіс балансу навантаження;

– інтегрувати кешування даних та результатів запитів;

– розгорнути систему з використанням хмарних провайдерів та контейнеризації.

База даних повинна бути децентралізованою, розмежована згідно до прав доступу, що надається користувачам системи в рамках тих задач, що вони повинні вирішувати. Оскільки в предметній області з часом можуть бути створені нові функції, сутності та може бути змінений підхід до навчання загалом, це має бути враховано (має бути реалізований CRUD на кожну з сутностей системи та інтерфейс для керуваннями ними адміністратором).

Up-time додатку не може бути меншим, ніж 99,9%, також він повинний витримувати навантаження не менше ніж 1000 користувачів в одиницю часу. Система матиме мікросервісну архітектуру та модель зв’язку RESTful API, як показано на рис. 2.1 [4].

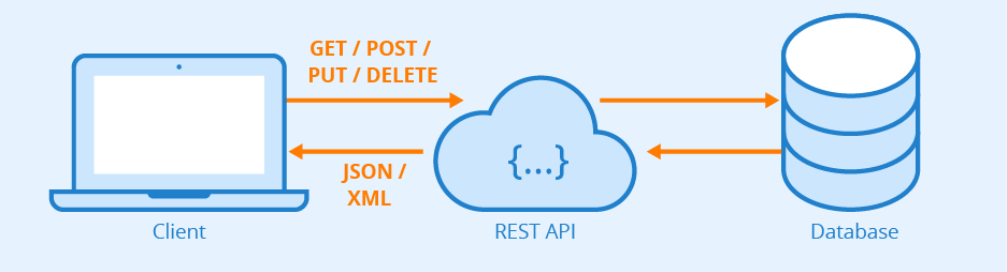

Рисунок 2.1 – Схема RESTful API

Основні технології:

– JavaScript – мова програмування, що відмінно працює як у браузері (на відміну від фавориту, що був до нього - PHP), так і на стороні серверу. Сучасний JS – це «безпечна» мова програмування, що спочатку була створена для браузерів. У браузері для JS є все, що пов'язано з маніпулюванням веб-сторінками, взаємодією з користувачем і веб-сервером [14];

– NodeJS для виконання скрипту на серверній частині. Це найбільш стабільне та прогресуюче рішення, що насправді є обгорткою над движком

виконання JavaScript – V8, який забезпечує роботу Google Chrome [8]. Платформа додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API, написаний на C++, підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим вводом/виводом. Фреймворком обрано Fastify - продуктивний та повністю асинхронний;

– React – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка спрямована вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими зустрічаються при розробці веб-додатків. React обробляє тільки користувацький інтерфейс у веб-додатках [14].

– Docker – програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux -систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами [9];

– MongoDB – документоорієнтована СУБД, що підходить для вирішення майже будь-яких задач. Вважається однією з класичних прикладів NoSQL-систем, використовує JSON-подібні документи і схему бази даних [10]. В даній роботі не було розгорнуто базу даних власноруч, замість цього використано вже готове хмарне рішення від розробників самої MongoDB, а саме - Atlas [11]. MongoDB Atlas працює на AWS, Microsoft Azure та Google Cloud Platform. Таким чином, вирішені питання масштабування системи на рівні БД (clustering, sharding), безпеки та відмовостійкості;

– кеш даних реалізовано з використанням Redis – резидентної СУБД класу NoSQL з відкритим вихідним кодом, що працює зі структурами даних типу

«ключ-значення» [12]. Використовується як для баз даних, так і для реалізації кешей, брокерів повідомлень. Орієнтована на досягнення максимальної продуктивності на атомарних операціях (заявляється про приблизно 100 тис. SET і GET запитів в секунду на Linux-серверах початкового рівня). Написана на C, інтерфейси доступу створені для більшості основних мов програмування. Також, використано npm модуль для реалізації кешу запитів lru;

– авторизація та аутентифікація реалізовані на базі JWT. JSON Web Token (JWT) – це відкритий стандарт (RFC 7519) для створення токенів доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи. Токен JWT складається з трьох частин: заголовка (header), корисного навантаження (payload) і підпису або даних шифрування [14];

– сервери розміщені на DigitalOcean [13]. Цей сервіс надає хмарні послуги для розробників, дає можливість розгортати і масштабувати додатки одночасно на декількох комп'ютерах.

2.2 Мікросервісна архітектура

Мікросервісна архітектура – варіант сервіс-орієнтованої архітектури програмного забезпечення, спрямований на взаємодію невеликих, слабо пов'язаних і легко змінюваних модулів - мікросервісів. Якщо в традиційних варіантах сервіс-орієнтованої архітектури модулі можуть бути досить складними програмними системами, а взаємодія між ними часто покладається на стандартизовані великовагові протоколи (такі, як SOAP, XML-RPC), в мікросервісній архітектурі системи складаються з компонентів, що виконують як елементарні функції, так і взаємодіючі з використанням економічних мережевих комунікаційних протоколів (в стилі REST з використанням, наприклад, JSON, Protocol Buffers, Thrift) (див. рис. 2.2). За рахунок підвищення гранулярності модулів архітектура націлена на зменшення ступеня зачеплення і

збільшення зв'язності, що дозволяє простіше додавати і змінювати функції в системі в будь-який час.

Властивості, характерні для мікросервісної архітектури:

– модулі можна легко замінити в будь-який час: акцент на простоту, незалежність розгортання та оновлення кожного з мікросервісів;

– модулі організовані навколо функцій: мікросервіс по можливості виконує тільки одну досить елементарну функцію;

– модулі можуть бути реалізовані з використанням різних мов програмування, фреймворків, сполучного програмного забезпечення, виконуватися в різних середовищах контейнеризації, віртуалізації, під керуванням різних операційних систем на різних апаратних платформах: пріоритет віддається на користь найбільшої ефективності для кожної конкретної функції, як стандартизації засобів розробки і виконання;

– архітектура симетрична, а не ієрархічна: залежності між мікросервісами однорангові.

Архітектурний стиль мікросервісів – це підхід, коли єдиний додаток будується як сукупність невеликих, самодостатніх, незалежних, не тісно пов'язаних сервісів, що спілкуються між собою за допомогою легких механізмів як HTTP, що використано в даній роботі, так і по gRPC чи AMQP. Ці сервіси побудовані навколо бізнес-потреб (кожен відповідальний за конкретний процес) та розгортаються незалежно з використанням повністю автоматизованого середовища. Одна з причин використання мікросервісів полягає в тому, що компанії хочуть мати можливість швидко щось змінювати, щоб швидко реагувати на зміни бізнес-вимог, випередити конкурентів.

Найбільш популярне середовище для виконання мікросервісів – системи управління контейнеризовано додатками (такі як Kubernetes та їх надбудови OpenShift і CloudFoundry, Docker Swarm, Apache Mesos), в цьому випадку кожен з мікросервісів, як правило, ізолюється в окремий контейнер або невелику групу контейнерів, доступну через мережу іншим мікросервісам і зовнішнім споживачам, і управляється середовищем оркестрації, що забезпечує

відмовостійкість і балансування навантаження. Типовою практикою є включення у контур середовища виконання системи безперервної інтеграції, що забезпечує автоматизацію поновлення і розгортання мікросервісів.

Мікросервіси допомагають розробникам доставляти зміни швидше, безпечніше і з більш високою якістю, тобто зберігати швидкість розвитку продукту, навіть коли той стає неосяжних розмірів. Моноліт - будується як єдине ціле. Будь які зміни, навіть самі невеликі, потребують перебудови та розгортання всього додатку. З часом стає складніше зберігати хорошу модульну структуру, зміни логіки одного модуля мають тенденцію впливати на код інших модулів. Монолітні програми також може бути важко масштабувати, коли різні модулі мають конфліктні вимоги до ресурсів.

Архітектура постійно піддається критиці з самого моменту її формування. Серед нових проблем, які виникають при її впровадженні зазначаються:

– мережеві затримки: якщо в модулях, що виконують кілька функцій, взаємодія локальна, то мікросервісна архітектура накладає вимогу атомізації модулів і взаємодії їх по мережі;

– формати повідомлень: відсутність стандартизації та необхідність узгодження форматів обміну фактично для кожної пари взаємодіючих мікросервісів призводить як до потенційних помилок, так і складнощів налагодження.

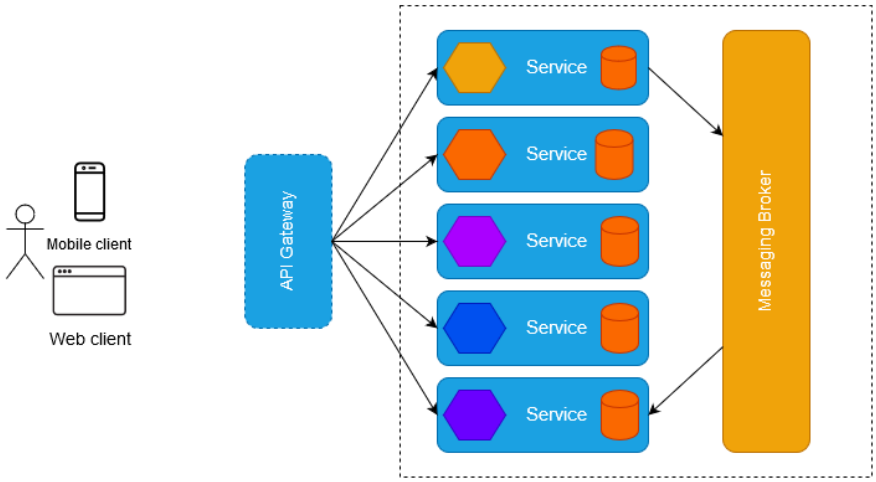

Рисунок 2.2 – Мікросервісна архітектура

3. АЛГОРИТМИ РОБОТИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

3.1 Алгоритми рекомендації контенту

Рекомендаційні системи – програми, які намагаються передбачити, які об'єкти (фільми, музика, книги, новини, веб-сайти) будуть цікаві користувачу, маючи певну інформацію його профілю.

Дві основні стратегії створення рекомендаційних систем – фільтрація на основі вмісту і колаборативна фільтрація. При фільтрації на основі змісту створюються профілі користувачів і об'єктів. Профілі користувачів можуть включати демографічну інформацію або відповіді на певний набір запитань. Профілі об'єктів можуть включати назви жанрів, імена акторів, імена виконавців та іншу атрибутивну інформацію в залежності від типу об'єкта. Наприклад, в Music Genome Project музичний аналітик оцінює кожну композицію за різними музичними характеристиками, які можуть використовуватися для виявлення музичних уподобань користувача. При колаборативній фільтрації використовується інформація про поведінку користувачів в минулому – наприклад, інформація про покупки або оцінки. В цьому випадку не має значення, з якими типами об'єктів ведеться робота, але при цьому можуть враховуватися неявні характеристики, які складно було б врахувати при створенні профілю. Основна проблема цього типу рекомендаційних систем - «холодний старт»: відсутність даних про нових в системі користувачів або об'єктів.

В процесі роботи рекомендаційні системи збирають дані про користувачів, використовуючи поєднання явних і неявних методів. Приклади явного збору даних:

– запит у користувача оцінки об'єкта за диференційованою шкалою;

– запит у користувача ранжування групи об'єктів від найкращого до найгіршого;

– пред'явлення користувачеві двох об'єктів з питанням про те, який з них кращий;

– пропозиція створити список об'єктів, улюблених користувачем.

Приклади неявного збору даних:

– спостереження за тим, що оглядає користувач в інтернет-магазинах або базах даних іншого типу;

– ведення записів про поведінку користувача онлайн;

– відстеження вмісту комп'ютера користувача.

Рекомендаційні системи порівнюють однотипні дані від різних людей і обчислюють список рекомендацій для конкретного користувача. Для обчислення рекомендацій використовується граф інтересів. Рекомендаційні системи - зручна альтернатива пошуковим алгоритмам, так як вони дозволяють виявити об'єкти, які не можуть бути знайдені останніми. Цікаво, що рекомендаційні системи часто використовують пошукові машини для індексації незвичайних даних.

Умовно всі існуючі системи рекомендації контенту на сайті поділяють на чотири типи:

– колаборативна фільтрація. Спочатку система визначає, що відомо про відвідувача, потім знаходить тих користувачів, які мають схожі інтереси для обчислення прогнозу. Після цього підбирає набір рекомендацій для конкретного користувача і ранжує їх на підставі даних про CTR сторінки або розділів майданчика. Недолік підходу: щоб працювати з цими фільтрами, потрібно постійно отримувати інформацію про те, як саме взаємодіє кожен користувач з сайтом;

– технологія, заснована на знаннях. Ці рекомендаційні системи засновані на знаннях про ту чи іншу предметну область (knowledge-based);

– системи, засновані на контенті (content-based) - більш узагальнена реалізація попередньої технології. Вона підбирає об'єкти за принципом подібності. На технології content-based сьогодні працюють не тільки сайти магазинів, але і такі проєкти як Prismatic і вітчизняний Surfingbird;

– рекомендаційні системи на основі гібридних технологій (hybrid) дуже важкі в розробці, проте одні з найрозумніших і точних. Тут використовуються найскладніші алгоритми, які постійно удосконалюються. В основі підходу

лежить ідея «взяти найкраще від кожного». Неважко здогадатися, що коштувати таке рішення буде дорого, а використовувати його під силу лише великим корпораціям типу Amazon і Netflix.

Найбільш доцільним алгоритмом надання рекомендацій, при розробці онлайн-системи навчання інформатики буде колаборативна фільтрація. При цьому існує 2 способи її використання: підхід, заснований на моделі, та підхід, заснований на пам'яті. Підхід на основі моделі добре працює при великій кількості даних, тому нам необхідно використання саме підходу на основі пам'яті. Для цього підходу запам'ятовується матриця корисності і рекомендації складаються шляхом запиту даного користувача до іншої частини матриці корисності. Розглянемо приклад: у нас є фільми *i* користувачі *U*, і ми хочемо дізнатися, наскільки користувач *i* любить жанр відео *k*.

$$\bar{y}_i = \frac{1}{|I_i|} \sum_{j \in I_i} y_{ij}$$

Рисунок 3.1 – Формула розрахунку середнього рейтингу

На рисунку 3.1 представлено формулу середнього рейтингу, який сформований для користувача на основі всіх відеороликів, які він коли-небудь оцінював. Використовуючи це, ми оцінюємо рейтинг матеріалу *k* для користувача *i*, як показано на рис. 3.2:

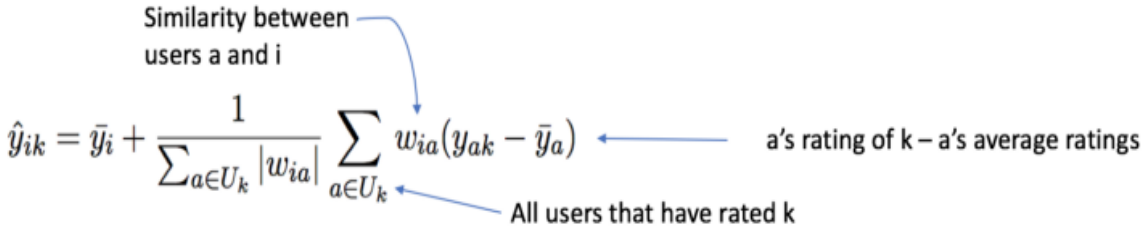

Рисунок 3.2 – Оцінка рейтингу для жанру *k*

Подібність між користувачами можна обчислити з використанням будь-яких методів, таких як косинусна схожість / схожість Жаккара / коефіцієнт кореляції Пірсона і т.д. Ці результати дуже легко отримувати та інтерпретувати, але як тільки дані стають занадто розрідженими, продуктивність погіршується.

4. РЕАЛІЗАЦІЯ ОНЛАЙН-СИСТЕМИ ВИВЧЕННЯ ІНФОРМАТИКИ

4.1 Архітектура проєкту

При проєктуванні додатку з вивчення інформатики виділено наступні сервіси (див. рис. 4.1):

– єдина точка входу - API Gateway, який бере на себе такі типові завдання з управління API, як безпека, кешування і моніторинг. Аутентифікація користувачів проходить саме в цьому сервісі. Система моніторингу завжди знає стан сервісів, що обробляють запити користувачів і таким чином, вирішує питання щодо перенаправлення запиту на виконання до якогось конкретного;

– API сервіс, що обробляє запити користувачів. Як і попередній - розроблений з використанням NodeJS + Fastify. Драйвером для роботи MongoDB є нативний модуль з npm - mongodb. Для підключення до Redis - ioredis. Реалізовано версіонування API (для мобільних додатків). Функціонал системи, що може працювати в фоновому режимі винесено до Realm системи в Atlas, такі ж операції, як формування тамбнейлів винесені з основного потоку роботи у дочірні за допомогою нативного модулю NodeJS – cluster;

– сервіс Front, що містить в собі сайт проєкту. Реалізовано з використанням TypeScript + React & Redux. Бандл та розміщення серверу всередині контейнеру забезпечує Webpack. Транспілятор - Babel. Це єдиний сервіс, в якому застосовано TypeScript через особливості розробки фронтенду. Загалом, використання TypeScript значно збільшує кількість часу на розробку, тому backend частина розроблена тільки на JS;

– Redis. Офіційний образ сервісу можна знайти на Docker Hub, лише для написання власної конфігурації. Кожен такий сервіс йде в парі з сервісом API.

Кожен сервіс є контейнером Docker, глобальні налаштування для розгортки системи прописані через docker-compose. Набори контейнерів розміщено на серверах з ОС Linux (Ubuntu 20.04), де, окрім розгортки сервісу та стандартних налаштувань:

– встановлюється firewall для контролю трафіку – ufw;

– встановлюється Nginx для забезпечення шифрування з'єднання, сертифікати отримані безкоштовно від Let's Encrypt - центру сертифікації, що надає безкоштовні криптографічні сертифікати X.509 для TLS-шифрування. Оновлення сертифікатів (період існування - 90 днів) проводиться автоматично за допомогою утиліти certboot.

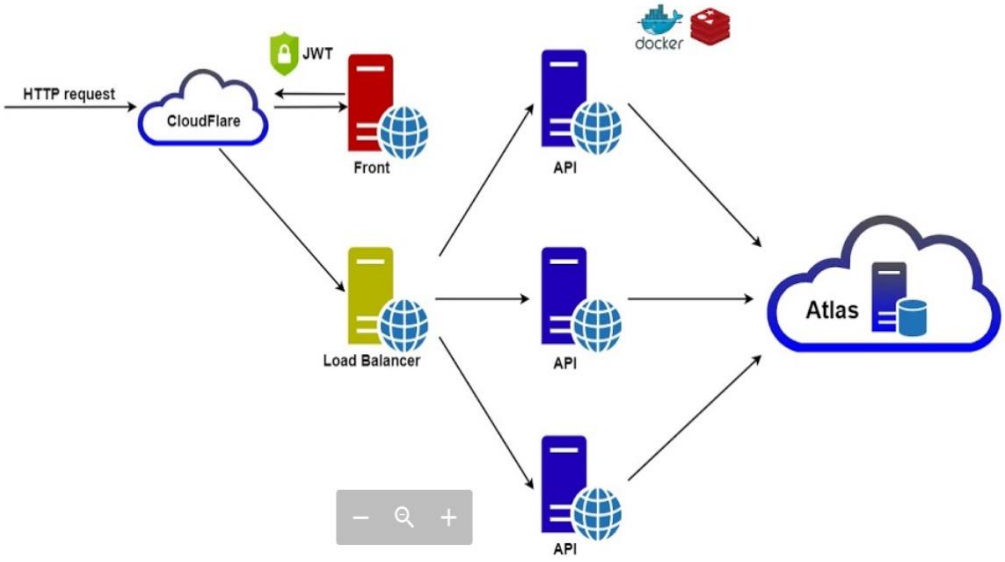

Рисунок 4.1 – Схема архітектури проєкту

Схема роботи аутентифікації:

– клієнт проходить аутентифікацію в додатку (наприклад, з використанням логіна і паролю);

– у разі успішної аутентифікації сервер надсилає клієнту access і refresh токени;

– при подальшому зверненні до сервера клієнт використовує access-токен. Сервер перевіряє токен на валідність і надає клієнту доступ до ресурсів;

– у разі, якщо access-токен стає дійсним, клієнт надсилає refresh-токен, у відповідь на який сервер надає два оновлених токена;

– у разі, якщо refresh-токен стає дійсним, клієнт знову повинен пройти процес аутентифікації.

JWT має ряд переваг над куки: при використанні куки сервер повинен зберігати інформацію про видані сесії, в той час як використання JWT не вимагає зберігання додаткових даних про видання токенів: все, що повинен

зробити сервер - це перевірити підпис. Сервер може не займатися створенням токенів, а надавати контент буде задачею інших модулів.

У JSON-токенах можна зберігати додаткову корисну інформацію про користувачів. Як наслідок – більш висока продуктивність. У разі з куки іноді необхідно здійснювати запити для отримання додаткової інформації. При використанні JWT ця інформація може бути передана в самому токені. JWT надає одночасний доступ до різних доменів і сервісів (див. рис. 4.2).

З точки зору безпеки, аутентифікація за допомогою JWT потребує збереження access-токену на клієнтській частині тільки в оперативній пам'яті (XSS типи атак), а refresh токен міститься в cookie з налаштуваннями httpOnly (не має доступу до вмісту файлу) та sameOrigin - прив'язка до домену сайту (захист від CSRF). Зокрема, менджером HTTP заголовків основною мірою виступає npm модуль helmet. Захист від DDoS атак реалізовано з використанням in-memory-cache (lru) та модулю, що запам'ятовує IP-запиту та встановлює ліміти на одиницю часу.

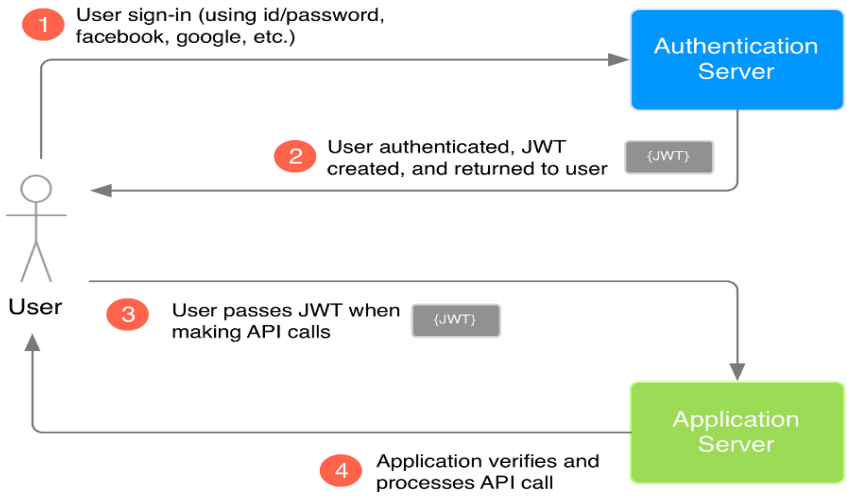

Рисунок 4.2 – Процесс аутентифікації JWT

Оскільки NodeJS – це платформа, що розроблена таким чином, щоб розробник міг обирати будь-яку архітектуру незалежно від поданого інструментарію. В різноманітних проєктах обирається найбільш доцільний формат згідно предметної області. Також слід зазначити, що проєкт було розроблено, використовуючи JavaScript більш як функціональну мову програмування, без використання класів. Таким чином, зменшується час

виконання коду (збільшується його швидкодія) та досягається менший час відгуку.

4.2 Програмна реалізація

При розробці проєкту створено 5 github репозиторіїв: api, balancer, front, infrastructure та Qa (див. рис. 4.3). Перші три з перерахованих слугують для розміщення коду відповідних сервісів, згідно до архітектури проєкту з п. 4.1. Infrastructure – репозиторій, що розміщує в собі конфігурації docker-контейнерів для локального запуску проєкту, налаштування та сховище для redis, сценарії оболонки для розгортання та застосування міграцій (див. рис. 4.6). Qa – невеликий сервіс, що слугує точкою надходження запитів при тестуванні контролерів API та відтворює поведінку сервісів, що будуть інтегруватися з зовнішнім API сервісом у майбутньому. Кожен сервіс містить внутрішню директорію deploy, що містить у собі налаштування docker, nginx та redis для розгортання проєкту в production режимі на окремому сервері для кожного з сервісів.

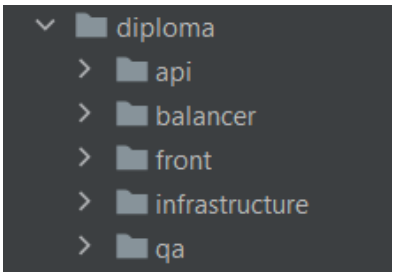

Рисунок 4.3 – Загальна структура коду проєкту

Сервіси api, balancer та qa розроблені за однаковим шаблоном, тому загальну структуру серверної частини буде продемонстровано на прикладі сервісу api, оскільки він є найбільшим серед усіх.

Складові частини API (див. рис. 4.4):

– config, директорія, що містить у собі конфігурації fastify серверу та плагінів, що були підключені до нього: cache, cookie, cors, jwt, multipart, static, swagger;

– controllers, директорія з контролерами роутів сервісу. В свою чергу розділена з врахуванням майбутнього версіонування та виділенням статичних контролерів;

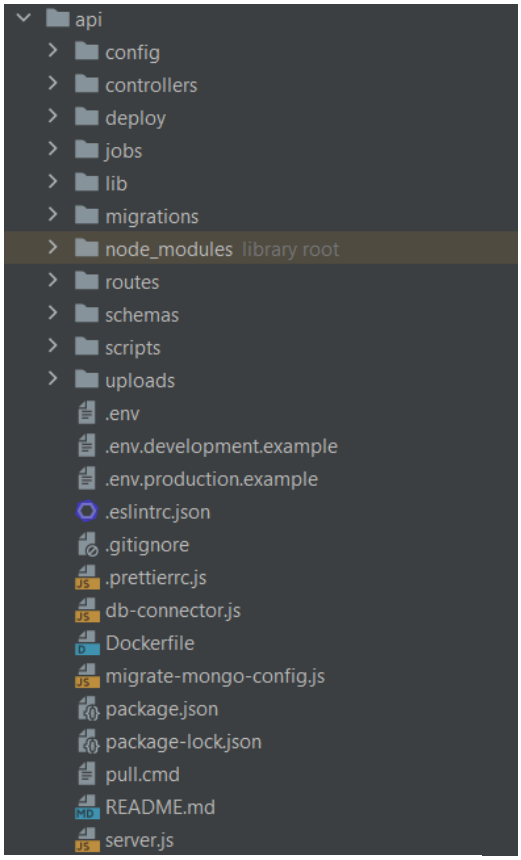

Рисунок 4.4 – Структура сервісу API

– deploy, директорія з налаштуваннями docker, nginx та redis для розгортання проєкту в production режимі. Також містить у собі декілька сценаріїв для автоматизацій розгортки;

– jobs, директорія з cron-завданнями, тобто скриптами, що запускаються сервером відповідно до заданих графіків;

– lib, директорія, що містить у собі допоміжні засоби (функції, обгортки над роботою зі сторонніми API, тощо), що не пов'язані з бізнес-логікою проєкту та можуть бути використані в будь-якій його частині. Кожна з експортованих функцій є константою, циклічні імпорти всередині модулю відсутні;

– migrations, директорія з міграціями бази даних;

– routes, директорія з роутерами сервісу. Також містить у собі функціонал кешування запитів;

– schemas, директорія з json-схемами. Розділена на 2 частини: схеми валідації операцій вставки та оновлення документу в колекції MongoDB (застосовуються при запуску екземпляру сервісу та зберігаються на рівні СУБД), та схеми валідації на вхідні дані для роутерів та вихідні дані для клієнтів. Схеми на вихідні дані сприяють пришвидшенню json під час відповіді, але додають умови до розробників: дані, що не відповідають схемі, не будуть відправлені користувачу;

– scripts, директорія з shell-скриптами. Ці сценарії використовуються декількома контролерами під час роботи сервісу, задля виконання їх в окремому процесі linux всередині docker-контейнера;

– uploads, директорія для збереження завантажених користувачами файлів. Ця директорія це volume між контейнером та локальною файловою системою серверу;

– файли .env, набори змінних оточення для кожного з етапів розробки сервісу: development та production;

– .eslintrc.json, .prettierrc.js, файли конфігурацій модулів, що призначені для збереження чистоти та якості коду;

– db-connector.js, сценарій підключення до кластеру Atlas;

– Dockerfile, сценарій побудови docker-контейнера;

– migrate-mongo-config.js, файл конфігурації міграцій проєкту;

– package.json, package-lock.json, залежності проєкту та лок файл для фіксування версій завантажених пакетів;

– server.js, точка входу сервісу.

Структура сервісу front побудована, спираючись на те, що використовується TypeScript та React.

Складові частини сервісу front (див. рис. 4.5):

– src, директорія з кодом веб-додатка (сайта), що транспілюється з TypeScript у JavaScript, створюється банд утилітою webpack та потім

надсилається усім користувачам системи. Функціонал для роботи з Redux розміщено у піддиректорії bus;

– typings, типи даних;

– .eslintrc.json, конфігурація лінтеру;

– Dockerfile, сценарій побудови docker-контейнеру;

– package.json, yarn-lock.json, залежності проєкту та лок файл для фіксування версій завантажених пакетів;

– server.js, точка входу сервісу;

– tsconfig.json, webpack.config.js, конфігурації файлів для компіляції TypeScript та створення бандлу відповідно.

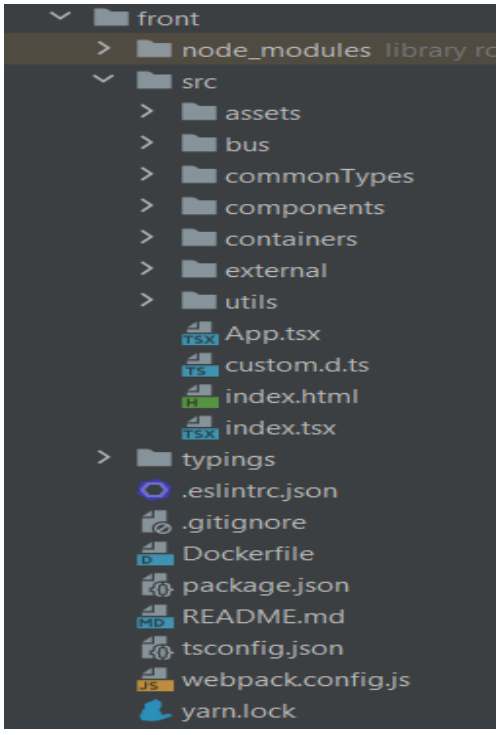

Рисунок 4.5 – Структура сервісу front

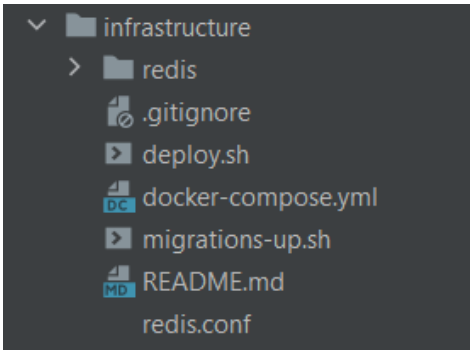

Рисунок 4.6 – Репозиторій infrastructure

ВИСНОВКИ

В рамках даної роботи проаналізовано варіанти проєктування та розробки веб-додатків на прикладі рекомендаційної онлайн-системи з вивчення інформатики. Основною метою було дослідження проблем розробки, знаходження рішень, що мають відповідати вимогам більшості веб-додатків.

З роботи можна зробити висновок, що більшість важливих операцій для розробників не залежать від бізнес-логіки проєкту, що буде написана в контролерах чи тригерах. Проєктування вагомо впливає на час розробки проєкту, більшість операцій ідентична, не залежно від розміру команди розробників чи проєкту. Відсутність одного чи декількох основних шаблонів при розробці на мові JavaScript та веб-додатків в цілому призводить до збільшення витрат компаній на їх створення, тому це питання в наш час є дуже важливим та все ще залишається відкритим.

Запропоновано стек технологій та загальний перелік питань, що має бути вирішений за для успішного запуску проєкту. Проаналізовано основні онлайн-системи навчання інформатики та обрано створення власної.

Використання матеріалу з даної роботи для розробки подібного додатку має наступні переваги:

– низька ціна розробки додатку;

– висока ефективність роботи, належний рівень відмовостійкості;

– прискорення виконання робіт через інтегрування зовнішніми сервісами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Теория и методика изучения информатики [Електронний ресурс] – Режим доступу: https://sites.google.com/site/methteachinfo/lec/lec-9.

2. Основные проблемы разработки современных интерфейсов [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/435912.

3. ТОП-10 платформ дистанционного обучения для школьников [Електронний ресурс] – Режим доступу: https://www.study.ru/article/sovety/top-10-online-shkol-dlya-shkolnikov

4. Архитектура веба [Електронний ресурс] – Режим доступу: https://tproger.ru/translations/web-architecture-101.

5. 5 основных проблем веб-дизайна и разработки [Електронний ресурс] – Режим доступу: https://blog.depositphotos.com/ru/veb-dizajn-i-razrabotka.html

6. Ричардсон Крис, Микросервисы. Паттерны разработки и рефакторинга [Електронний ресурс] – Режим доступу: https://www.piter.com/product/mikroservisy-patterny-razrabotki-refaktoringa.

7. Введение в REST API — RESTful веб-сервисы [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/483202.

8. Timur Shemsedinov, Современный курс по Node.js в 2020 [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/485294.

9. Docker documentation [Електронний ресурс] – Режим доступу: https://docs.docker.com.

10. MongoDB documentation [Електронний ресурс] – Режим доступу: https://docs.mongodb.com/manual.

11. Atlas documentation [Електронний ресурс] – Режим доступу: https://docs.atlas.mongodb.com.

12. Redis documentation [Електронний ресурс] – Режим доступу: https://redis.io/documentation.

13. DigitalOcean – The developer cloud. Deploy and scale seamlessly [Електронний ресурс] – Режим доступу: https://www.digitalocean.com.

14. Современные архитектуры фронт-энда [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/500072.

15. Защита Nginx с помощью Let's Encrypt в Ubuntu 20.04 [Електронний ресурс] – Режим доступу: https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04-ru.

16. ouTube's Recommendation Engine: Explained [Електронний ресурс] – Режим доступу: https://hackernoon.com/youtubes-recommendation-engine-explained-40j83183.